CS112 - Fall 2022
Project01 - Blackjack
Instructor: Paul Haskell

INTRODUCTION
For Project01 you can choose to develop a program that plays Blackjack. At the end of the project, we will have a Casino Day in which everyone's programs play a tournament together, with a single Blackjack Dealer coordinating the play. Grading is not based on the outcome of the Blackjack tournament (maybe we play more than one), but rather on code quality and sophistication of your approach to planning the game.

Your game is not an interface for a person to play Blackjack. Your program will make the playing decisions, playing against another computer program that acts as the Blackjack dealer. Your player and the dealer may play hundreds of hands per second!

Before the actual Casino Day, you will have an opportunity to test your program in class with a simplified Dealer.

Blackjack Review
Blackjack, also called "21", is a card game in which Players make bets, receive cards, and make plays hoping to win hands often enough to accumulate a big stack of money.

Betting
Each Player starts with $500 in her account. The game is played through multiple hands. The Player places a bet before each hand, before seeing any of the cards dealt in the hand.

Dealing
Each hand, after betting, the Dealer deals two cards to each player and to herself. All of the Players' cards are face-up (i.e. visible to all). One of the Dealer's cards is face-up and the other is face-down. The game is played with standard 52-card playing decks: Spades, Hearts, Clubs, Diamonds, and card values of Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King. For this tournament, the Dealer will shuffle and deal from a "shoe" containing <u>seven</u> combined 52-card decks (total of 364 cards).

Scoring your Hand
The goal of each hand of Blackjack is for you to get a higher score than the Dealer, while keeping your score at 21 or under. If your score goes above 21, you "bust" and lose the hand.

- Each number card ("2" through "10") has a score equal to its numerical value.
- Each face card (Jack, Queen, King) is worth 10 points.
- Each Ace is worth either 1 or 11, whichever gives a more favorable score. Scores over 21 are unfavorable. A hand in which an Ace is valued at 11 is called "soft", because the total of the hand can be reduced later if necessary, by reducing the Ace value to 1.

Playing a Hand
As a Player, when it is your turn, you have a choice of several "plays":

- **Stand**: stop getting any more cards. Wait to see if your score for this hand beats the Dealer.
- **Hit**: get another card from the Dealer and add the card to your hand.
- **Double down**: after each hand is dealt, before hitting or standing, the Player can "double down". This means the Player doubles her bet and gets exactly one more card, with no more ability to make any more plays in the current hand.
- **Split**: after each hand is dealt, before hitting or standing, if the Player's two cards form a pair (the cards must have equal value, so for example a Jack and a Queen counts as a pair), then the Player can split the pair. In this case, the Player's pair is separated into two separate hands. The Dealer places an additional card onto each original card, so the two hands each have two cards. And the Player must match her original bet on the new hand. The Player then plays each hand in turn. If one (or both) of the Player's new hands is again a pair, then the Player can split again, if she has enough money for the bet on the new hand. (If not enough money, then she cannot split.)
- In a hand, a Player can keep hitting (unless she doubled down) until she busts or she decides to stand.

Winning and Losing a Hand

If the first two cards in a hand are an Ace and a 10-value card, the hand has a value of 21 before any play begins. This is called a Blackjack.

If a Player has a Blackjack, then she wins 150% of her bet from the Dealer, plus her original bet back, unless the Dealer also has a Blackjack (a tie, called a "push"), in which case the Player only gets her original bet back.

If a Player does not have a Blackjack but the Dealer does have a Blackjack, then the Player loses the hand, and her bet is forfeit to the Dealer.

If neither a Player nor the Dealer has a Blackjack:

If a Player hits too long and her hand's score goes over 21, she loses the hand, and her bet is forfeit to the Dealer. If a Player stops hitting before going over 21, then the Dealer plays.

Dealer play is pretty simple: the Dealer must hit until her score is 17 or higher. Then she must stand. (The dealer does not hit soft 17's.)

After the Player and Dealer have played, if the Player has a higher score than the Dealer, then the Player gets her original bet back and the Dealer also pays the Player an amount equal to the bet.

If the Player and Dealer have the same score (a push), the Player's bet is returned.

If the Dealer busts and the Player did not, the Player gets her bet back and the Dealer also pays the Player an amount equal to the bet. Even if Dealer busts, if the Player busts first, she forfeits her bet to the Dealer.

If the Dealer has a higher score than the Player and did not bust, then the Player forfeits her bet to the Dealer.

Each hand, all Players play in parallel against the Dealer. In a hand, some Players may win, some may tie the Dealer, and some may lose.

Each Player plays multiple hands until she runs out of money. Then she is out of the game. For the Casino Day tournament, the winner is the person who runs out of money last.

Your Program

The instructor has written a Java program that will perform as Dealer. Each of your Player programs will communicate with the Dealer over the classroom WiFi network, using the Internet Protocol ("IP"). Luckily, Java makes it easy to set up this communication.

Communication with Sockets

Your **Blackjack.java** program should take two command-line arguments, both Strings:

- IpAddress: IP network of Dealer
- IpPort: IP port of server

And your program should include something similar to the following code:

```
import java.io.Socket;
import java.io.IOException;
import java.io.DataInputStream;
import java.io.DataOutputStream;

Socket socket = new Socket(IpAddress, Integer.valueOf(IpPort));
DataInputStream dis =
        new DataInputStream(socket.getInputStream());
DataOutputStream dos =
        new DataOutputStream(socket.getOutputStream());
```

Your program will read commands from the Dealer by reading dis, and it will write data to the Dealer by writing dos. Here is code for reading and writing:

```
private void write(String s) throws IOException {
        m_dos.writeUTF(s);
        m_dos.flush();
}
private String read() throws IOException {
        return m_dis.readUTF();
}
```

Game Protocol

A <u>protocol</u> is a set of rules by which two (or more) entities interact with each other. Your program will communicate with the Dealer by implementing the following protocol. The Dealer will initiate communications every time, by sending a command (along with data) to your program. Your program should repeatedly try to read from the Dealer. When a command is received, your program interprets it and responds. Most commands require a reply. Here are the Dealer commands and required replies:

- **login** – whenever your program receives this command, it must reply with
  `<<your GitHubId>>:<<your avatar name>>`
  Your program may receive the "login" command more than once. You always must reply with the same values. Your "avatar" name will be shown on a graphics window, to update everyone on the Casino Day game status. Your GitHubId will be known only to the instructor.
- **bet:<<your current amount of money>>:all:<<list of *all* cards that have been played in this and previous hands since the cards were last shuffled>>** - you must reply with:
  `bet:<<amount of your bet for this hand>>`
  The amount you bet in this current hand must be at least $1 and at most your current amount of money. If you try to bet more than you have, that is cheating and you are kicked out of the game. Do not include the '$' symbol in your reply, just give an integer. You can use the list of all cards played to tailor your bet and your playing strategy to maximize your chances of winning.
- **play:dealer:<<dealer "up" card>>:you:<<list of all of *your* cards in this hand>>** - this command gives the current status of your current hand, and requires that you reply with one of the following:
  ```
  hit
  stand
  double
  split
  ```
  If you double or split after hitting, double or split when you do not have enough money to cover the cost, or split when you do not have a pair, then you are cheating and you are kicked out of the game.
- **status:<<win** or **lose** or **push>>:dealer:<<dealer score>>:you:<<your score>>** - you should not reply to this message. You can print it out, parse it for use playing the game, etc. The score may be "blackjack" or a number. There are two alternative forms of the "status" message:
  status:lose:you:<<your score>> - if you bust
  status:win:you:blackjack – if you get a Blackjack and the Dealer does not
- **done:<<message>>** – you should not reply to this message. You are done with the game, either because you ran out of money or because you broke the rules. The `message` will explain why. You can close the `socket` and exit the program.

Details

- As you see, commands and replies often use the colon character ":" as a separator.
- Some of the commands contain a card or list of cards. A card is represented with two or three characters. The last character is the "suit": S for spades, H for hearts, C for clubs, D for diamonds. The first character is the card value: 2-9, J for jack, Q for queen, K for king, A for Ace. Only the 10 requires three characters: "10" plus the suit. Examples are:
  2C

AD
10S
QH

In lists of cards, the cards are separated by ":".

- It is a requirement that your program respond to every command within 1 second, or else your program will be deemed nonresponsive, and you will be kicked out of the game.

Strategy for Play

There is a LOT of information on the Internet about how to play Blackjack effectively. After you get your basic program working, you can search to see what strategies (1) make sense to you, and (2) you want to incorporate into your program.

Here are some basics:

- If your score is 11 or lower, you can hit safely at least once without fear of busting. Even if you are dealt an Ace, the Ace will count as 1.
- If your score is close to but less than 21, it is risky to hit. But you should look at what card the Dealer has showing and decide what total the Dealer might have.
- In Blackjack, many of the cards are worth 10. If you are dealt a pair of Aces, it probably makes sense to split them, in hopes of getting one or two hands with value 21.
- Similarly, if your opening hand has value 11, it almost surely makes sense to double down. You have a good chance of getting a ten-value card and winning with **double** your original bet.
- In real casinos, "card counting" i.e. keeping track of all the cards that have been played, is considered cheating. Here, it is encouraged, and as noted above, the Dealer in fact tells you all cards that have been played since the cards were last shuffled. You can use this information to make smart decisions about how to bet and how to play.
- For example, the rules of Blackjack are such that the Dealer wins slightly more than 50% of the time, due to the fact that if both the Player and Dealer bust, the Player still loses her bet. However, the Player wins 150% of her bet if she gets a Blackjack. If the probability of a Blackjack is high (i.e. lots of Aces and ten-value cards remain in the deck), then a bigger than normal bet might make sense.

Grading

Your program's performance in the Casino Day tournament does not affect your grade—it is just for fun. Your deliveries for this project must come in two parts

- Part 1 supports Internet connectivity and legal response to commands
- Part 2 implements your best algorithms for betting and for playing

For grading, after Part 2, you must schedule a 10-minute time slot with the instructor to present your code to the instructor and/or TA's. During this meeting, you will explain your strategy for playing the game and will give a walkthrough of your code. Your code will go through an automatic tester beforehand (separate from the class Casino Day) to see how well it followed the rules of the Project.

Put your program into a subdirectory called "**Project01**" inside your MyWork directory, and remember to push your Project01 to GitHub before the deadlines.

- Part 1 must be turned in before Weds October 19
- Part 2 must be turned in before Weds October 26

Rubric

Milestone	Points	Comments
Part 1: Program is in correct location, pushed to GitHub before Part 1 deadline, compiles successfully, and connects to Dealer server	15	
Part 1: correct play: program gives correct reply messages, plays correctly in given test situations	35	Response must be given within 1 second Always bets $1 Always replies to "play" command with "stand" Closes socket after "done" command
Part 2: correct play: program gives correct reply messages, plays correctly in given test situations	70	Response must be given within 1 second Never bets more money than available Never splits or doubles after hitting Never splits a non-pair Never splits or doubles when not enough money left
Part 2: Intelligent rules for play	50	Auto grader tests multiple game scenarios
Part 2: Intelligent rules for play that take advantage of card counting	20	Auto grader tests multiple game scenarios
Part 2: Software quality	20	Judged subjectively by graders
10-minute interview shows understanding of one's own software. Bring an informal one-page document describing how you tested your program	40	Judged subjectively by graders

Conclusion

I hope this project ends up being fun. You will implement a communication protocol, your program will communicate with other computers, you will get to explore game strategy, and to decide what strategy to implement.

This project is an excellent opportunity for you to practice testing! You might want to build a simple Dealer to deal random cards to your Player, to see which playing strategies do well and which do poorly. You will certainly want to test to ensure that your program implements the game protocol correctly.